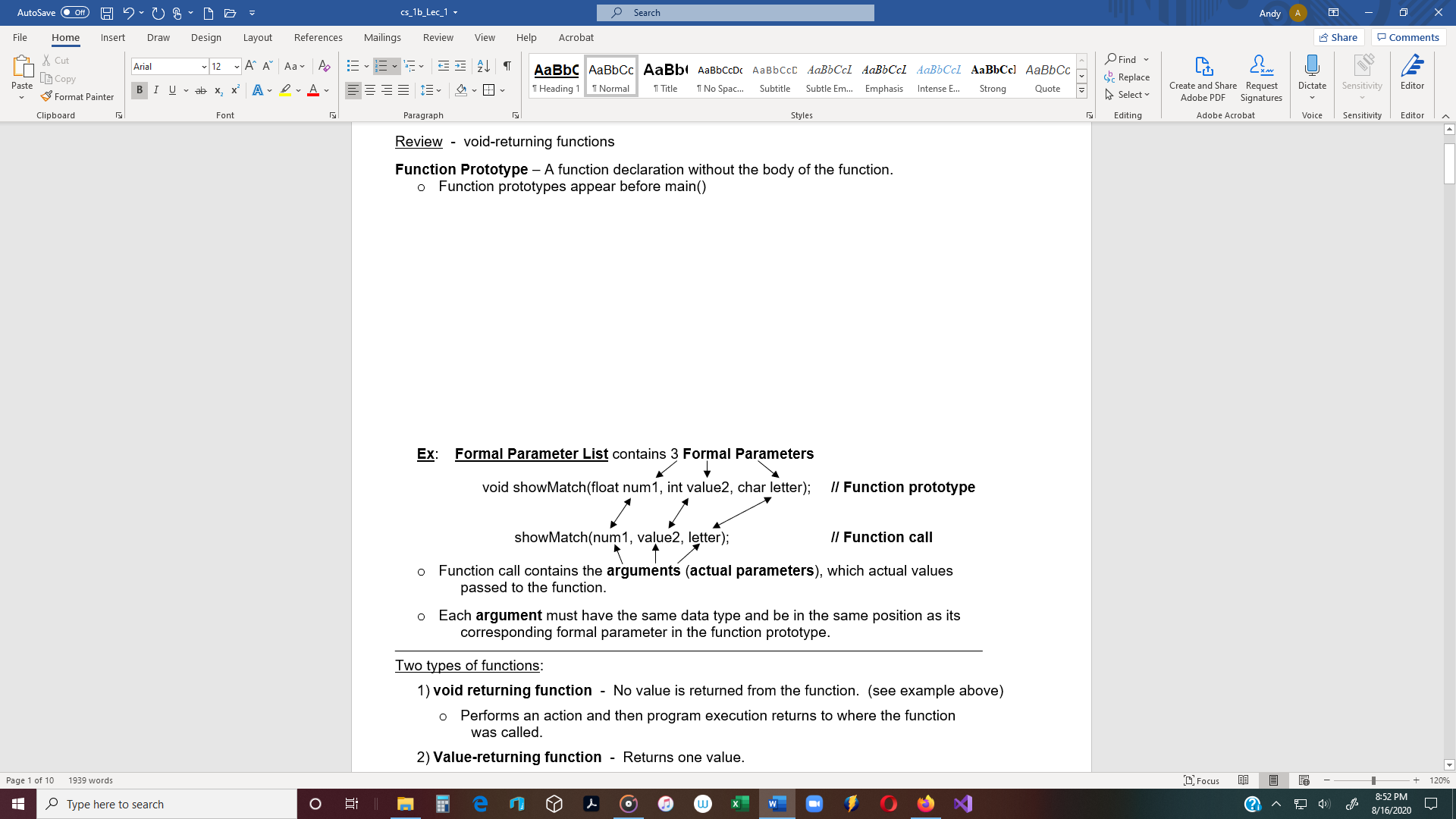
# Lecture #1 – Functions - void-returning functions - value-returning functions

Review - void-returning functions

**Function Prototype** – A function declaration without the body of the function.

* Function prototypes appear before main()



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Two types of functions:

* 1. **void returning function** - No value is returned from the function. (see example above)
* Performs an action and then program execution returns to where the function

was called.

* 1. **Value-returning function** - Returns one value.
* This type of function calculates a value, or gets a value from a user or file, and

then returns to where it was called.

**Value-returning Functions** - When a function returns a value, the function call must be

in an expression.

Ex:float calcPmt(float loanAmt, float rate);// function prototype

int main()

{

// Function call - 2 arguments

are passed to the function.

float loatAmt = 1000;

float rate = .07;

monthlyPmt = calcPymt (loanAmt, rate);

// A float value is returned to where the function was called, and assigned to the variable **monthlyPmt**.

**Function definition** – The code that defines what the function does. (The actual function)

* Function implementations are listed after main(), after the closing curly bracket.
* Function definitions can appear in any order.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Example of 2 function calls:

// Function Prototypes

int getAge();

void displayAge(int age);

int main()

{ No parameters (nothing is sent to the function)

int age; (Notice the prototype has no parameters.)

age = getAge( ); 🡨 (1.) **getAge() function call** - The call is in an

assignment statement because a value is being

returned. (value-returning function)

One actual parameter (argument) is sent to the function.

(Notice that the prototype has one parameter.)

displayAge(age); 🡨 (2.) **displayAnswer() function call** - The call cannot

be in an assignment statement because no value

value is being returned. (void-returning function)

return 0;

} // end of main()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ex: Function implementation - (value-returning function)

int getAge() 🡨 **function header**

{

int age;

cout << “Enter your age: ”;

cin >> age;

return age; 🡨 NOTE: Value-returning functions always have a **return** statement

} // end of getAge()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ex: Function implementation - (void-returning function)

void displayAge(int age) 🡨 **function header**

{

if (age >= 21)

cout << “You can drink.”;

else

cout << “Not today.” // void-returning methods do not a return statement

} // end of displayAnswer()

**Calling a Function**

Ex: **Value-returning function:**

* In the following example, the function calcSquare is called. It performs a math operation and returns an integer value to main.

**Note**: A value-returning function can only return **one value.**

#include <iostream>

using namespace std; // Includes the identifiers with <iostream & fstream.

int calcSquare (int number1); // **Function prototype**

int main()

{

int number = 5;

int result;

result = calcSquare(number); // **Function call** - The **argument** (5) is passed to the

// function. The function calculates the square of 5

// and returns 25. The 25 is then assigned to result.

cout << “The square of ” << number << “ is ” << result << endl;

return 0; // return 0 means that a value of 0 is returned to the caller.

}

// See next page for the function implementation of the calcSquare() function.

// The following is a function that calculates and returns the square of a number.

// The function receives the argument (5), evaluates the expression, & returns an integer value.

**Function implementation** is the entire function heading and body.

(also called **function definition**).

int calcSquare( int number) // **Function heading**

{

int result;

Function

Body

result = number \* number

return result;

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Formal Parameter** – Items declared within parentheses in a function prototype, as well as,

in a function heading.

* + Formal parameters in a function prototype are not variables.
  + The purpose of a function prototype, is to give information to the compiler about a function

that may be called in the program.

* + Function prototypes are not executable code.

Important: Formal parameters in a function heading become variables when the function

is called.

**Actual Parameter** – When a function is called, a value, called an actual parameter is passed

(sent) to the function, to be used by the function.

* + **Argument** - Actual parameters are often called arguments.
  + More than one argument can be passed.
  + The data type of each argument in a function call must match a corresponding parameter

in the function prototype.

**Note**: When a function is called space is allocated in memory for the function’s formal

parameters and local variables.

**Pass By Reference** - Passing arguments by reference to get more than one value.

**Parameter Passing** – Two kinds of parameters.

1. **Value Parameter –** The called function receives a copy of the argument’s value.
   * When a function returns, actual parameters and local function variables are destroyed.
   * This is called **Pass by Value**.
2. **Reference Parameter –** The called function receives a copy of the argument’s memory

address (not the value of the argument).

**Pass by reference** - When arguments are passed by reference, the arguments are

memory addresses of variables declared in main().

**Pass by reference** – Include an ampersand ( **&** ) in front of the variable in the

function prototype and function heading.

**&** - **“address”** operator

**Ex:** #include <iostream>

using namespace std;

void printData(int temp, string month); // Pass by value

void getData(int **&** temp, string **&** month); // Pass by reference

int main()

{

int temp;

string month;

getData(temp, month); // The function call is the same whether or

printData(temp, month); // not it is pass by reference or value.

return 0;

}

// ================================================================

// ==== **getData** ===================================================

void getData(int **&** temp, string **&** month) // Pass by reference (include &)

{

cout << “Enter the month: “;

cin >> month;

cout << “Enter the temperature: “;

cin >> temp;

}

// ================================================================

// ==== **printData** ==================================================

void printData(int temp, string month) **//** Pass by value

{

cout << “On this day in ” << month << “ the temperature is: ”

<< temp << “.\n”;

}

// ================================================================

**Coding Style Guidelines**

* + The Coding Style Guidelines provide guidelines and examples of good coding style.
  + The following guidelines are established internationally accepted rules for writing and

formatting C++ code.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Information Header** - Must have an information header at the top of each homework

assignment.

Example:

// Attached: HW\_2a, 2b, and 2c 🡨 If there are 3 parts to this assignment, they should be

// listed at the top and **stapled together**.

// ===========================================================

// File: HW-2a

// ===========================================================

// Programmer: Tom Lee

// Class: CMPR 121

// Instructor: Dennis Rainey

//

// Description:

// This program calculates the **…….**

// **……………………………………..**

// ===========================================================

1. **Variables Names**
   1. Variable names should give information about the variable.
   2. Variable names start with an lower case character.
   3. The first letter of the second word should be uppercase.

Ex: age firstName bal amt

* 1. Constant variables should be in all uppercase, and an underscore should be

used if two words.

Ex: const float RATE = 0.08;

Ex: const int LEGAL\_AGE = 21;

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Function Names**
   1. Function names should be verb-oriented.
   2. Function names start with a lower case character.
   3. The first letter of the second word should be uppercase.

Ex: calcSum print getName readFile

1. **Indentation** **and Spacing**
   * + All code must be indented and spaced appropriately.
     + One space after **if**, **switch**, **for**, and **while**

Ex: if (……) switch (……) for (……) while (……)

(1 space) (1 space) (1 space) (1 space)

* + - No space between a function name and the parenthesis

Ex: calcSum(int num1, int num2) print(int sum)

(No space) (No space)

* + - No more than one blank line in code, unless for separating functions.
      * **Functions** - 3 blank lines separate function implementations.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5.) **Separator Lines** =======================

- Two lines above main

- Separator lines above and below every function header.

- Separator line after each function implementation.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6.) **Spacing Function Information Header** - Each function header with the following:

- Function name - The name should be in the top separator line.

- Description - Describe what the function does.

- Input - A description of the function’s input.

* + - * Each formal parameter should be named and a description of its data type

and purpose should follow.

- Output - A description of the function’s output

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

7.) **Spacing around operators** - Put one space on each side of an operator.

- C++ has many operators: Insertion << and extraction >> operators

Comparison operators: >, >=, <, <=, ==, etc

Math operators: \*, /, %, +, -

Logical operators: !, &&, ||

Ex\_1: if (num1 **>** num2)

1 space on each side of the operator

Ex\_2: if (num1 **>** num2)

Ex\_3: cout << num2;

1 space on each side of the operator

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

8.) **Spacing Comments**

- Comments should be placed immediately above code being described.

- Comments should be left justified with the current block

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Example of C++ Coding Style**

// Attached: HW\_2a, 2b

// ===============================================================

// File: HW\_2a.cpp

// ===============================================================

// Programmer: Jim Smith

// Class: CIS 17A

// Instructor: Dennis Rainey

//

// Description:

// This program is a contrived example to illustrate the coding style

// guidelines handout. The user is prompted for a positive integer, which

// is used as an upper bound for a loop. Then the sum of all integers from

// one to the upper bound is calculated and displayed to the standard

// output stream.

// ===============================================================

#include <iostream>

using namespace std;

int calcSum(int limit);

bool testIntValue(int value);

// ==== main =====================================================

//

// ===============================================================

int main()

{

bool isValid;

int sum;

int upperBound;

// get the upper bound from the user

cout << "Please enter a positive integer value: ";

cin >> upperBound;

// make sure the value is valid

IsValid = testIntValue(upperBound);

if (false == isValid)

{

cout << "Sorry, you must enter a positive value\n";

exit(EXIT\_FAILURE);

}

// calculate the sum from one to the upper bound

sum = calcSum(upperBound);

cout << "The sum is " << sum << endl;

return 0;

} // end of main()

// ===============================================================

// ===============================================================

(**3 blank spaces separates main and the function**)

// ==== calcSum ==================================================

// This function calculates the sum from one to the limit parameter

// (inclusive), then returns the value to the caller.

//

// Input:

// limit - An integer value, representing the upper limit,

// is passed to the function.

// Output:

// An int value, representing the sum of all integers, from one

// to the limit parameter, is calculated and returned.

// ===============================================================

int calcSum(int limit)

{

int counter;

int total;

for (counter = 1, total = 0; counter <= limit; ++counter)

total = total + counter;

return total;

} // end of calcSum()

// ===============================================================

(**3 blank spaces separate funtions**)

// ==== testIntValue =============================================

// This function tests the input value to make sure that it is

// positive. If it is, a value of "true" is returned; else, a

// value of "false" is returned.

//

// Input:

// value - An int value to be tested is passed to the function.

// Output:

// The integer value is tested. A value of true is returned

// if the input is positive. Otherwise false is returned.

// ===============================================================

bool testIntValue(int value)

{

// if the input is less than zero, return false, else return true

if (value < 0)

return false;

else

return true;

} // end of testIntValue()

// ===============================================================